

**JPL D-48347**

***Jupiter Europa Orbiter Mission***  
**ASIC via FPGA Guidelines**  
with Addendum on Europa ASIC Process Flow

Prepared by

Dr. Gary R. Burke  
Jet Propulsion Laboratory  
4800 Oak Grove Drive  
Pasadena, CA 91109

October 7, 2008

Version 1.1

*For Public Release*

Copyright ©2008 by the California Institute of Technology  
Government Sponsorship Acknowledged



Jet Propulsion Laboratory  
California Institute of Technology  
Pasadena, California

## Table of Contents

1.0 Scope .....	4
2.0 Testability .....	4
2.1 Test Coverage .....	5
2.2 Scan Path .....	5
2.3 BIST .....	7
2.4 Special tests.....	8
2.5 Parametric test – inputs .....	8
2.6 Parametric test – outputs .....	8
2.7 Boundary scan.....	8
2.8 AC test logic .....	9
3.0 Test vector Generation .....	9
3.1 Automated Test Pattern Generator (ATPG).....	10
3.2 IDDQ testing.....	10
4.0 Additional Design Issues.....	10
4.1 Digital Clock Manager (DCM).....	10
4.2 RAM .....	11
4.3 Clocks .....	11
4.4 IPs .....	11
4.5 Partitioning .....	12
4.6 Initialization .....	12
4.7 Timing .....	12
4.8 Analog Circuitry .....	13
4.9 Control/monitor Registers and Multifunctions .....	13
5.0 Packaging.....	14
6.0 Timing Closure .....	14
7.0 Tools .....	14
7.1 Synthesis .....	15
7.2 Static Timing Analysis .....	16
7.3 Final Netlist Check.....	16
8.0 Summary of FPGA/ASIC differences .....	17
9.0 References .....	19
Appendices .....	19
A1: Yield .....	19
A2: Structural testing of ASICs.....	20
A3: Types of ASIC .....	21
A3.1: Gate Arrays .....	21
A3.2: Standard cells .....	21
A3.3: Structured arrays .....	21
A3.4: Via programmable.....	22

Addendum .....	23
----------------	----

## **1.0 Scope**

The current plan for the proposed JEO mission is to use ASICs to meet complex digital logic requirements instead of using FPGAs. This is driven by the extreme radiation environment surrounding Jupiter. FPGAs play an important role in the successful development of ASICs however. The verification and validation process of an ASIC is significantly enhanced by the use of FPGAs during development. The FPGAs can be programmed early on with the final or intermediate design and extensively tested on breadboards and in the system. It is recommended that such a combined process is used with the design started on an FPGA and finishing on an ASIC. An addendum is attached at the end of this document to address ASIC process flow for Europa.

## **2.0 Testability**

Testability is the major difference between ASIC design and FPGA design. Improving FPGA testability to meet ASIC requirements has many design implications, which are listed in this section.

Testability is needed in ASICs to ensure fabricated chips and packaged parts can be sorted into good and bad parts. The semiconductor process does not yield 100% good parts, and so these need to be sorted before packaging. Problems in packaging can also destroy a part, so the packaged parts also need to be tested. In the case of FPGAs, this testing is done by the manufacturer, before user configuration, and so is transparent to the user. For ASICs the testing is performed by the manufacturer or a specialized 'test house', using test vectors supplied by the user, and is after the user configuration

The chips are approximately 500mils square and are manufactured on a silicon wafer of up to 12 inches in diameter. Additional process monitors (resistors, transistors) are routinely placed on the wafer to help the manufacturer check the process is within tolerance. Defects on the wafer, contaminants, mask misalignments or process variations across the wafer conspire to produce a yield (Y) which is less than 100%. A process is characterized by 'defect density' (D), usually given as defects per sq cm. More information on yield is in Appendix A1.

Testing after manufacturing is performed with a Tester. These are large expensive machines which are capable of supplying sequences to all input signal pins and recording and checking the output from all output pins. More information on testers and yield issues is found in Appendix A2.

## **2.1 Test Coverage**

A physical defect can result in 1 or more faults in the ASIC.

To eliminate parts with faults requires more than functional testing. The reason is that the fault may have only a subtle effect, which is not exercised in the functional tests. It is necessary to test the chip structurally. Structural testing will check a high percentage of possible faults.

There are many kinds of defects. But to simplify test generation a 'stuck at fault' model is used that has been shown to be effective.

The fault coverage (or test coverage)  $T$  is the percentage of faults that are potentially detected by the test vectors. This needs to be as high as possible or there will be test escapes, i.e. parts that are placed in service which actually have defects.

The formula for calculating test escapes (see A1) depends on both the test coverage  $T$  and the Yield  $Y$ . As an example a process with a defect density of 0.01 might yield over 80% for a die size of 4 cm<sup>2</sup>. With test coverage of 90%, the test escapes will be 2%. This reduces to 1% if the test coverage is 95%.

This is even more apparent in a fabrication process with lower yield (such as a rad-hard foundry). A yield of 10% will give a test escape of 20%, with 90% test coverage. We would need test coverage of 99% to bring the test escapes down to 2%.

The 2% figure means there is an 87% chance that 1 in 100 parts will have defects after testing good. For 10 parts with 2% yield, the probability of at least one bad part is 18%. This is clearly unacceptable, and shows the need for high test coverage.

## **2.2 Scan Path**

It is difficult to achieve even 90% fault coverage. Each potential stuck at fault needs to be reached in 2 ways:

1. It needs to be controlled, i.e. set to 1 and 0
2. The effect needs to be observed, i.e. the effect of the fault has to eventually affect an output.

This is impossible to achieve with conventional functional testing on an unmodified FPGA design.

To improve test coverage, the design needs to include various dedicated circuits. In particular, it needs to include a scan path. The scan path is a giant shift register, made up of most flip/flops in the design. Ideally every flip/flop is included but in practice some have to be left out (see below). The idea is that by scanning in a pattern to every flip/flop a desired state is reached (control). Subsequently clocking the circuit once, causes the resultant state to be set in the same scan chain. By then scanning out, the effect of the potential fault is determined (observed).

It is impractical to insert the scan path in the source code, and typically the synthesis tool e.g. Synopsis will insert the scan chain by substituting all flip/flops with 'scan flip/flops/' and hooking them up.

However this is only feasible if certain rules are used:

1. All flip/flops are normally clocked using the posedge of the clock only (or all negedge). This is recommended for other reasons e.g. to simplify timing analysis. If there is a mixture of posedge and negedge clocks then separate scan paths need be created. If there are flip-flops not on a regular clock (i.e. clocked by a signal), then the scan path cannot be created. There are other good reasons for disallowing such logic as well, since it disobeys the basic rules of synchronous logic.
2. All flip/flops use the same clock; or provision is made in the design to insert a scan clock into the various clock trees. In this case care must be taken to ensure low slew on these gated clocks.
3. Existing shift registers in the design should have scan controls added separately to avoid duplicating scan. These then need to be 'black-boxed' for scan insertion.
4. Tri-states in the design must be designed to avoid conflicts during scan. This is usually done by gating off tristate controls during scan.
5. The output of the scan flip/flops should not be connected to any asynchronous set/reset input of any flip/flop. This is best achieved by ensuring asynchronous set/reset is only used for startup POR reset.
6. Scan control circuit and ports (test mode) needs to be added to put the circuit into scan mode. Also scan-in and scan-out ports need to be added.
7. There should be no redundant logic. Redundant logic is logic which is not actually needed to give the correct result. This logic makes it impossible for 'Automated Test Pattern Generator' (ATPG) to setup tests for nodes affected nodes, resulting in low fault coverage.

Traditionally, redundant logic is inserted by the designer in error. However, with the recommended design approach which uses HDL language such as Verilog and synthesis to generate the gate level logic, inadvertent redundant logic is eliminated.

However, redundant logic may be deliberately inserted in a design to improve fault tolerance. A well-known example is triple-module-redundancy (TMR). TMR will defeat ATPG, since a fault in TMR logic will by design not affect an output. There are various techniques for solving this. The simplest is to ensure that the outputs of TMR logic are registered separately before being combined. These registers of course need to be on the scan path.

### **2.3 BIST**

Some embedded circuits such as RAMs are not readily amenable to scan testing. They are also difficult to test functionally unless they can be easily accessed from the I/O ports. This is not usually the case.

The preferred method of testing embedded RAMs is using Built In Self Test (BIST). The BIST is a circuit added to the design when enabled will locally test a RAM and report back the test results to the user. Each RAM will have its own BIST.

The BIST consists of 2 parts

1. A pattern generator
2. A signature circuit

The pattern generator will

1. Generate addresses – usually every address is checked so this a counter
2. Generate data patterns which can be written into the RAM
3. These patterns will be e.g.
  - Walking 1's or 0's
  - Special patterns to exercise known fault mechanisms
4. Control read and write cycles on the RAM

The signature circuit will take the RAM outputs and accumulate them, such that a failure will give a change in the result.

During test, the BIST is clocked many times and then the signature register is read out using scan. Note that self correction circuitry such as EDAC (error detection and correction) must be disabled during this test or bypassed by the signature register.

## **2.4 Special tests**

Custom circuits such as analog blocks will require specialized testing, with or without the scan path. If scan path is used, it is used to bring the chip to the right mode to test these circuits and the digital inputs to these circuits, then dedicated I/Os are used to provide the analog inputs and outputs needed. A 'system clock' and 'scan out' can then be used to observe the outputs of these circuits.

For generating a range of digital inputs a BIST circuit can be used similar to the RAM BIST.

## **2.5 Parametric test – inputs**

As part of the manufacturers wafer test and package test procedure, the manufacturer normally needs to test input thresholds. Since this is difficult to test using the existing user logic, the user inserts an 'AND tree' that couples an output to multiple inputs. Every input is chained to the AND tree which eventually is connected to an output. In this way each input can be individually toggled by the tester, while the remaining inputs are held all high or low. The effect of this input while it is ramping slowly up or down is observed on the output pin of the AND tree.

## **2.6 Parametric test – outputs**

Similarly the manufacturer requires testing of output levels. In order to get good test coverage, every output is toggled during the execution of the test vectors, but it is more conveniently monitored if the outputs are individually toggled. Although, it is simple to put the outputs on the scan path, we need to avoid this approach as this will duplicate logic and violates good design practice. Instead, the JTAG boundary scan provides this functionality already and is a better way to toggle the outputs.

## **2.7 Boundary scan**

Boundary scan is traditionally used to check I/O connections between components on a board. The standard for boundary scan is JTAG.

In general JPL boards will only have limited use for JTAG testing, since they contain only a small number of parts which have boundary scan. But JTAG is used for other purposes. For example, it can be used to initiate BIST.

JTAG comes with every FPGA, but needs to be added for ASICs. Due to its standard nature, the manufacturer will usually provide a recommended procedure to add JTAG, which involves

- Selecting JTAG cells for all I/O
- Hooking up boundary scan chain

- Adding JTAG controller (aka TAP controller)
- Add boundary scan inputs and outputs

## **2.8 AC test logic**

Usually the test sequence is run at low speed to avoid many problems (see A2). It is necessary to check that the ASIC does meet the speed requirements. To do this a small sequence of test vectors is used, which measures the delay through a certain long path on the ASIC. Since critical paths are usually embedded deep in the logic, the best way to handle this is to provide a dedicated path for this test. This is a delay chain consisting of a large number of invertors. It requires an input pin and an output pin. However, these can use the scan chain input and output, selected by the test mode inputs. Since the output buffer delay depends on output loading and tends to dominate the delay chain measurement, the delay chain can be divided such that 2 delay measurements can be made. These are subtracted to cancel the I/O delay. Again the 2 paths can be selected using the test mode inputs.

One may also insert a mode where the internal delay chain loops back to itself. This results in a self-oscillation, the frequency of which is related to the internal delay. This can be easily measured on a scope as an extra check after delivery. For unambiguous oscillation, a prime number of delay stages is used for the delay chain.

## **3.0 Test vector Generation**

Functional Test vectors can be produced during Verilog simulation. A sequence is run, and the sampled inputs and outputs collected using display statements. It is important to choose a good sample point – this is sometimes different for inputs and outputs. The resulting table can be formatted for the use of the Tester, using a script or c-code program.

The problem with functional vectors is that they produce low fault coverage. It is unlikely to be more than 80%. Additionally it is difficult to determine the fault coverage. The designer needs to run a ‘fault simulation’ which is very time consuming.

A better approach is to use ATPG to generate the bulk of the fault coverage, and then a few additional functional vectors for trouble spots such as the scan path itself.

### **3.1 Automated Test Pattern Generator (ATPG)**

Software will generate scan vectors and will indicate coverage as the program progresses. Initially, high coverage is obtained with a few vectors, but the rate of return drops exponentially. The last 10% of fault coverage may be impossible to obtain in a reasonable amount of time. The tools will generate a list of faults which have not been tested. The designer can use this list to create some additional vectors which will exercise logic which is inaccessible to the ATPG. For example, faults in the scan chain and scan logic need to be tested this way.

### **3.2 IDDQ testing**

This is very similar to scan path testing. It uses ATPG to generate vectors, but doesn't use the output signals (s) for observability. Fault observing is done by monitoring supply current  $I_{dd}$ . The concept is that a stuck at fault which is being exercised will result in a current which is slightly more than the default current.

IDDQ testing will give higher fault coverage than regular scan testing, with less test vectors.

However this relies upon:

1. Using CMOS logic which has virtually 0 quiescent current
2. Turning off devices on the chip which draw quiescent current, e.g.
  - RAMs
  - Digital Clock Managers (DCM)
  - Analog circuits

Although this worked well for earlier generations of CMOS, the newer 150nm and better processes have considerable leakage, which makes IDDQ testing a statistical process.

## **4.0 Additional Design Issues**

Generally, the high level Verilog code written for an FPGA can be retargeted for an ASIC during synthesis. There are a few problems which have to be dealt with:

### **4.1 Digital Clock Manager (DCM)**

DCMs are clock management blocks inside an FPGA. They can contain Delay locked loops, PLL, clock multipliers and clock dividers. They can be used to reduce skew between internal and external clocks and between internal clocks. They can also be used to generate different clock frequencies, either higher or lower than the original FPGA clock.

The issue is that these blocks are not usually available for ASICs. If a design needs a DCM function, then this may need a custom circuit to be generated.

## **4.2 RAM**

RAMs are used extensively in the new generation of FPGAs. They are also readily available on ASICs. The issue is that they are not generally directly compatible. The FPGA RAM is usually 2-port and supports only synchronous Read and Write operations. This may be different on the ASIC. For example, the ASIC RAM may be single port and asynchronous Read. The design would need to be modified to accommodate the RAM differences, for example cycling the RAM per operation. The other issue is RAM timing, which also tends to be different. For example, the access time of an FPGA RAM is 1ns on some FPGAs, but around 10ns on a rad-hard ASIC. This may need to be fixed in the design by including an extra pipeline register and associated control logic.

## **4.3 Clocks**

Besides DCMs (see 4.1) there are other clock issues. The clock buffering scheme is almost certain to be different on the ASIC compared to the FPGA. At the minimum this will involve instantiating clock buffers at the top level of the code. This may also involve separating logic into clock zones and instantiating separate clock trees. Additional buffer logic may need to be added on short paths between clock tree branch zones.

## **4.4 IPs**

The FPGA manufacturer provides a set of generated code (IPs) which can be freely used on its own FPGA. However this is not directly transferable to an ASIC.

- The source is usually not available
- The IP would need to be licensed from the FPGA manufacturer

In most cases these IPs will need to be written or purchased from an independent supplier. As well as the FPGA generated IPs, other IPs may have been purchased to facilitate the FPGA design. The license agreement will need to be renegotiated to allow the use of these on an ASIC.

In some cases the FPGA contains hard macros which are embedded in the design fabric such as RAMs and DCMs already. Other examples of IP blocks are

- Multipliers
- DSP blocks (multipliers adders)
- Processors (power pc)

For efficiency, these blocks will need to be written for the ASIC, and instantiated in the HDL Verilog.

#### **4.5 Partitioning**

Floorplanning is very important in an ASIC. Since the planning is performed on large related blocks of logic, it is important that the hierarchical module structure of the HDL reflect the desired placement. If the structure is badly written, a good floorplan may be difficult or impossible to achieve. For example, a RAM might be instantiated in a module which has the RAM buffers. But the RAM itself is a large item which should be placed independently. It would be preferable to have the RAM in its own module, and the RAM buffers in a separate module which can be placed near the ram i/o ports.

#### **4.6 Initialization**

The ASIC must be capable of being easily initialized to a known state. Without this the Tester results can be ambiguous.

A good FPGA design will already have this feature. Ideally all flip/flops are initialized by a POR circuit (this is itself synchronized - see guidelines Ref 1) independent of the clock.

#### **4.7 Timing**

The FPGAs are manufactured in a more aggressive technology than rad-hard ASICs. This means that the underlying technology (transistors) of the FPGAs is much faster than the ASICs. However, the programmable logic circuits and connections of an FPGA slow down the FPGA considerably. This more than counters the speeds advantage of the more advanced technology node. For example, a 250nm ASIC may be clocked significantly faster than the 90nm FPGA based prototype. The increase in speed can cause problems with hold time violations which were not present in the FPGA. This can be avoided by minimizing clock skew (see 4.3) and by adding buffers to short paths which may be effected. The buffers can be added automatically by the synthesis tool if this option is selected.

The faster ASIC timing simplifies some of the FPGA's more difficult logic. For example, it is difficult to get the PCI interface to work on an FPGA to spec, as timing requirements are so tight. To meet PCI timing, critical paths have to be hand compiled and hand placed. However, the automatic synthesis and 'Place and

Route' will work fine on an ASIC implementation, with just the usual floorplanning constraints.

Another timing value which is not a problem on FPGAs is 'transition time'. This is the rise and fall time of internal signals. A signal which is excessively loaded either by fanout or routing, can have an excessively high rise and/or fall time. The problem with high transition times is:

- it will increase the power as it takes so long to pass through the threshold
- timing calculations become very inaccurate as they depend on the exact threshold value which varies.

The solution is to constrain transition time in the synthesis tool setup. See also Section 6.0 for timing closure.

#### ***4.8 Analog Circuitry***

FPGA manufacturers start to include a small amount of analog circuitry in FPGAs. If this is used, it will need to be mapped onto an ASIC based board. Mixed mode (analog and digital) ASICs are possible, but not recommended because:

- digital process is not ideal for analog circuits
- difficult to test a mixed mode device, since analog testing is very different
- noise on digital power rails means the analog and digital power must be kept completely separate.

In general it is better to use standard analog parts with a digital ASIC, or a separate analog ASIC.

#### ***4.9 Control/monitor Registers and Multifunctions***

Both the FPGA and ASIC usually require registers which can be set or read under external control. An ID register will enable system software to determine the version of the FPGA or ASIC, and determine whether it is up to date or possible modify the control of the FPGA/ASIC. Other registers may set the FPGA/ASIC into various operating modes, to cope with slightly different applications or different connected equipment. For example if the same FPGA/ASIC is required to drive two different kinds of Pyro, a register value can select a different pulse width. Alternatively 2 or more different FPGAs /ASICs can be created.

Because of the higher NRE of ASICs, it makes more sense to use the same ASIC for several different functions rather than create a family of ASICs. The various functionalities can be determined by pins on the device and/or control registers.

## 5.0 Packaging

An ASIC gives a wider choice of packaging than an FPGA. If appropriate, a custom package can be purchased from another vendor. The package can be optimized for flight e.g.

- excellent heat path to substrate
- compact but reliable connections e.g. CGA
- hermetic sealing with no outgassing

A thermal analysis will need to be performed on a non-standard package.

## 6.0 Timing Closure

This is much more significant in ASICs than FPGAs. In FPGAs static timing analysis is performed by the designer using the vendor's tools, and it includes back annotated values from Place and Route. In the case of ASICs, these back annotated values are not available until late in the design process, since they are available only after place and route which is performed by the vendor. The designer needs to rely on front annotated values, which can be inaccurate.

Before manufacture it is necessary to update these delays with the actual delays extracted from the layout. These are the back annotated timing values. The designer uses these values to run static timing analysis and also to run some critical simulations to prove the timing is correct. For ASICs a good static timing analyzer is 'Primetime'. This tool will also calculate transition times.

## 7.0 Tools

The following tool chart shows the differences between ASIC tool and FPGA tools

TOOL	ASIC	FPGA
Simulation	Modelsim	Modelsim
Synthesis	Synopsis DC	Synplify
Test Insertion	Synopsis test	N/A
Static Timing Analysis	Primetime	FPGA Vendor Tools
ATPG	Fast-scan	N/A
Floorplanning	ASIC vendor tool	FPGA Vendor Tools
Place and Route	Cadence or Synopsis *	FPGA Vendor Tools
Parameter Extraction	Dracula *	N/A
Final netlist check	Formal verification tool	N/A
Layout Integrity	DRC *	N/A

Layout Integrity	ERC *	N/A
Layout vs netlist	LVS *	N/A

The tools marked \* are usually run by the vendor or at the vendors facility.  
The result of parameter Extraction is a timing file which is used by the Designer for Back Annotated simulation.

DRC is a design rule check on the layout. These design rules can be very complex in 150nm (and below) technology.

ERC is an electrical rules check on the layout, and LVS is comparing layout and 'schematic'. The schematic in this case is the synthesis output converted to a spice like netlist.

## 7.1 Synthesis

Although both FPGAs and ASICs need to be synthesized; synthesis is more complex on an ASIC, as there are more options to specify. In general, a different synthesis tool is used for ASICs, e.g. Synopsys, than for FPGAs (e.g. Synplify). For ASICs it is better to complete the Synthesis hierarchically. The reason is that different constraints may apply to different modules. It is typical to generate a script to run synthesis, which synthesizes the underlying logic and finally the top level.

Options that need to be considered for synthesis include:

- Advanced timing constraints such as multicycle, false paths, I/O timing for each module
- Test options such as type of scan path and how it is connected
- Rad-hard level of flip/flops in the module. The vendor usually provides a choice of flip/flop types. For critical logic a high level of radiation resistance may be required, whereas in a different module a lower level of radiation resistance can be used with subsequent area , timing and power savings
- Hold time fix logic. This option lets the synthesis tool add additional buffers where there is a potential hold time issue
- Fanout limit. This limit will help to improve performance of a block with long logic paths.
- Transition time limit. Described in 4.7
- State machine controls such as encoding methods to be used.
- Timing model choice to use for Front Annotated timing estimation.

## **7.2 Static Timing Analysis**

There are many more ways to violate timing on an ASIC than an FPGA. For example, a poorly balanced clock tree can cause excessive slew, resulting in hold time violations. Timing problems can occur due to temperature process and voltage variations. Additionally, the effect of radiation and aging need to be taken into account. The recommendation is to run static timing analysis at three conditions , worse case typical and best case, and to include the effects of aging and radiation.

## **7.3 Final Netlist Check**

With an ASIC the netlist may be changed directly after synthesis. This is (almost) never the case with an FPGA. For example, additional buffers may be added to fix a timing or transition time problem, clock tree routing may be changed to balance clock loading, I/O instances may be modified to add testability.

To verify the netlist still represents the original Verilog RTL design, formal verification can be used. This logically compares the final netlist with the original source. It extracts states and compares them. Usually some user interaction is required since signal names may have changed. It is a powerful tool which ensures the fabricated part matches the original HDL code. Back annotated simulation on the netlist also will prove the netlist is correct, but this is very time consuming, and Formal Verification will find subtle differences which may not be apparent in simulation.

## 8.0 Summary of FPGA/ASIC differences

	FPGA	ASIC	Reason
1	ideally all flip/flops clocked with same clock edge but mixed possible	all flip/flops clocked with same clock edge	allows connection to same scanpath. Otherwise see rule 3
2	ideally all flip/flops clocked with same clock. But multiple clocks possible	all flip/flops use same clock	simplifies scan clock insertion . If not possible use method 3
3	No scan logic	logic for scan clock insertion into all clocks	For multiple clock designs
4	No scan logic	Low skew in scan path clocks	in order to shift correctly scan clock needs low skew. Especially critical if clocks are gates e.g. when using multiple clocks
5	Tristates not recommended	No internal tristates in design or see rule 6	tristates can be activated during scan resulting in conflicts
6	Tristates not recommended	If tristates are in the design, logic needs to be added to ensure no conflicts during scan.	tristates can be activated during scan resulting in conflicts
7	Recommend flip/flop set/reset only used for POR	flip/flop set/reset only used for POR (or see 8)	This is to ensure scan-in or scan out operations do not asynchronously affect state of scan path during scan operation
8	no scan path	Any logic on asynchronous set/reset must not be connected to scan path even indirectly.	This is to ensure scan-in or scan out operations do not asynchronously affect state of scan path during scan operation
9	TMR used as needed	No redundant logic (or see 10)	ATPG cannot create test vectors for redundant logic
10	TMR used as needed	Redundant logic must be arranged to be tested independently, i.e. with redundancy removed.	ATPG cannot create test vectors for redundant logic
11	No special RAM test	Add BIST logic to test RAMs	RAMs cannot be easily tested using the scan path
12	no special functions	Add BIST Logic to test special functions	Custom blocks will need their own BIST
13	No AND tree	add AND tree for input cells	Needed for tester to test input thresholds

14	JTAG comes with FPGA part	add jtag circuit for output tests and others	Needed for tester to test output levels
15	No performance monitor	add delay chain to test gate speed	needed to simplify 'ac' test
16	No IDDQ logic	add logic to unpower RAMs and analog blocks	needed to IDDQ testing
17	Use DCMs if available	Replace DCMs if possible or modify to use available ASIC PLLs . If this is impossible then see 18	not usually standard
18	Use DCMs if available	Create or Find custom PLL	expensive but might be needed
19	Use FPGA clock buffers	Modify clock tree to fit available ASIC clock buffers	ASIC scheme will be different
20	Use core gen to simplify design	Remove IPs or purchase licenses for using the IPs on ASICs	FPGA free IPs are not transferable
21	No floorplanning	Repartition design hierarchy to match expected floorplan	Floorplanning not usually needed in FPGAs
22	Recommend every flip/flop initialized	Modify POR and initialization circuit	ASIC needs comprehensive initialization for consistent testing
23	Standard I/O and package	Develop pad ring and Package if needed	Full custom ASIC will need these
24	Use FPGA tools for timing closure	Need to use FA and BA values in stages	Timing closure depends on final layout
25	Limited multifunction use	More extensive multifunction use	High NRE makes a multifunction design more cost effective

## 9.0 References

- 1: 68892 Local Guideline [FPGA/ASIC Hardware Development, Rev. 0](#) Feb 04, 2005 [Develop Hardware Products Process Gary Burke](#)
- 2: 68532 Local Procedure [FPGA/ASIC Hardware Development, Rev. 0](#) Feb 04, 2005 [Develop Hardware Products Process Randel Blue](#)

## Appendices

### **A1: Yield**

For random defects the yield (Y) is given by the Poisson formula

$$Y = e^{-D*A}$$

Where A is the area of the chip and Y varies between 0 and 1.0.

In practice, the defect density does not have a random distribution, and defects tend to 'clump'. Various other yield models are used to account for this, the most used is Stapper

$$Y = (1/(1+(A*D/\alpha)))^{-\alpha}$$

Where alpha is selected to fit empirical data (0.3 to 5), and represents the amount of fault clustering.

Test Escapes  $T_E$  are the fraction of parts which were tested as good but actually are bad.

$T_E$  depends on both the yield, and the test coverage (T).

The following formula relates these:

$$T_E = 1 - Y^{(1-T)}$$

Where  $T_E$  is the test escape (0-1) i.e. bad parts divided by total parts

And T is test coverage (0-1)

The probability of a used part being bad (P) is a function of the total number of parts in use N, and  $T_E$

$$P = 1 - (1 - T_E)^N$$

## ***A2: Structural testing of ASICs***

Testing is performed both at the wafer level and at the packaged part. A Tester (e.g. Schlumberger, Teradyne) has a head unit which can either probe the pads of a chip on a wafer or has a socket for the packaged part. A load card is used for wafer probe. This is customized for the ASIC and contains multiple pins which are designed to land exactly on the pads around the chip. In the case of an ASIC gate array, a standard load card can be used. The tester runs a set of test vectors that are supplied by the chip designer. The designer produces these using ATPG and by running functional simulations and capturing the state of the I/Os.

One issue is how to deal with tri-state outputs. Generally these need to be listed as both inputs and outputs. When used as inputs the listed output state is X (don't care). In order for the tester to be in the right mode it will need a control signal. This signal can be an internal signal, usually the tristate enable, since it is not actually monitored during test. It is just used to switch the Tester channels from input to output. There are tricky timing requirements on this signal, since it takes time for the Tester to make this switch.



### **A3: Types of ASIC**

ASICs differ from FPGAs in that FPGAs are configured by the user after manufacture by the user, whereas ASICs are configured during the manufacturing process. Exactly how they are configured and the extent of configuring determines the type of ASIC.

#### **A3.1: Gate Arrays**

These are the closest in architecture to FPGAs. The manufacturer makes a master slice, where a regular array of transistors are already predefined. These transistor arrays are grouped into cells (which will become logic gates) that are geometrically aligned into columns across the die. Routing channels are defined in columns between cells, but not filled in. Pads are defined for I/Os. Power routing and clock buffers are predefined.

The customization is performed on the manufacturing floor during the backend metallization. The bottom 1 or 2 metal layers complete the gate definition, while the top layers define the interconnects.

One problem with a Gate Array is that it is difficult to include blocks such as RAMs. These need to be added to the masterslice itself, as they involve transistor placement and specialized routing.

Gate arrays are much cheaper than true custom ASICs, because the initial transistor array is mass produced and stored for future customization by the end user. The cost of the very expensive front end mask set is shared across a large number of parts. Furthermore, gate arrays have a shorter manufacturing cycle as pre-processed wafers are customized within 4 weeks or less. The primary disadvantage of gate arrays is that they are not as power efficient, highly integrated or fast as true ASICs.

#### **A3.2: Standard cells**

All silicon process and metal layers are customized, so RAM placement in the array is easier. Again cells are laid out in columns with routing channels just like a gate array. However there is more flexibility. Routing channel size can be adjusted to match the number of tracks needed. Performance is better than gate arrays for the same technology. A disadvantage is higher NRE cost, due to the larger number of custom masks required.

#### **A3.3: Structured arrays**

These are new types of ASIC. They combine the benefits of gate arrays and standard cells. Blocks of RAM and even a processor are predefined. Customization

is performed by metal layers hooking up these blocks with a gate array type area for random logic.

#### **A3.4: Via programmable**

This is a new type of ASIC, newer than structures arrays. All layers are including metal layers are predefined, but the connections between metal layers (vias) are programmable.

This is similar to a fuse programmable FPGA, but instead of fuses the last one or two metal layer are used to connect/program the array through vias.

The concept is also combined with structured arrays (see A3.3)

The Via programmable array combines many of the ASIC advantages including radiation hardness, with the low cost and short customization time of FPGAs.

The disadvantage is that the performance and density is be less than a custom ASIC, although better than an FPGA.

# **Addendum**

## **Europa ASIC Process Flow**

Gary R Burke

9/19/08

### **Background**

The proposed JEO mission must withstand the large radiation trapped in the Jupiter magnetic field. This has been estimated at ~2.9 MRads behind 100mils shielding. Current and projected FPGAs do not survive beyond 300K rad Therefore, the custom logic needs to be implemented in an ASIC. There are several ASIC foundries which allow use up to 1Mrad.

### **ASIC Advantages**

FPGAs have been preferred over ASICs in recent missions, since they are easier to modify and debug, and the NRE cost is much less. However, ASICs have some advantages over FPGAs:

- Able to withstand higher radiation total dose (TID) up to 1 Mrad, possibly higher.
- Built in mitigation for SEU effects using hardened flip/flops and memories.
- Less production cost per unit (although higher NRE).
- Higher performance and integration level
- Possibility to include custom analog circuits
- Easier to tailor to application (e.g. use of large amounts of RAM)

### **Existing process**

The 2 JPL local FPGA Procedure documents (ref 1, 2) describe the process of and the guidelines for designing FPGAs and ASICs. The process describes the various steps including the reviews needed, both formal and peer, and the documentation and reports which need to be delivered at each step. Since FPGAs are much more capable now, it is recommended that any ASIC designs start with a FPGA design. This can be thoroughly verified before committing to an ASIC.

## **Europa process**

In the combined process, (FIG. 1), the process is modified to include a FPGA design stage followed by an ASIC design stage.

A full version of this process is shown in FIG. 2.

The initial concept design applies to both the FPGA and ASIC stages. This is where the specification is written, which describes the functionality of the ASIC, the method of implementation, as well as the description of the proposed intermediate FPGA.

The FPGA design needs to be designed with the ASIC in mind. A comprehensive guideline to the FPGA design requirements is in a separate document (ref 1).

The ‘FPGA design and test’ step will complete the FPGA design using the guidelines, and test it by

- simulation
- breadboard

Since this is an intermediate step, the FPGA does not have to be rad-hard, and in fact a ram-programmable FPGA, such as the Virtex 4, is recommended. The final design needs to be synthesized for the ASIC and analysis performed on the ASIC design, to show the FPGA is on track with the requirements of the ASIC.

In the next step the FPGA design is converted to the ASIC design. This involves several changes, detailed in the preceding document (ref 3). The ASIC design is comprehensively tested in simulation. Test Vectors are produced to test the fabricated ASIC.

Since the ASIC device is not available at this time, the FPGA is updated and tested both at the card level and system tested. This ensures the final ASIC delivery has an excellent chance of first time success.

The ASIC is ‘place and routed’ and after several verification steps (detailed in fig 2) masks are made and the ASIC is fabricated. Fabrication time varies but is typically 3 to 6 months.

Finally the ASIC wafers are tested and sorted. The good parts are packaged and re-tested. Once the parts are delivered they are placed on EM then FM boards and retested, both bench test and system test.

Bugs at this time are unlikely since the design has already been checked out on the FPGA. Minor problems (e.g. pinout) may need to be fixed with a board or software change; more fundamental issues (e.g. non-functional RAM) may need another ASIC iteration.

### **Detailed Process**

The Process detailed in Fig 2 will be fully described in a future document.

The color coding is as follows:

Blue: FPGA design and test

Black: ASIC Design and Test

Red: Steps normally performed at the ASIC foundry.

### **References**

- 1: 68892 Local Guideline [FPGA/ASIC Hardware Development, Rev. 0](#) Feb 04, 2005 [Develop Hardware Products Process Gary Burke](#)
- 2: 68532 Local Procedure [FPGA/ASIC Hardware Development, Rev. 0](#) Feb 04, 2005 [Develop Hardware Products Process Randel Blue](#)
- 3: ASIC-via-FPGA-guidelines V1 10/17/08

FIG 1: Starting with FPGAs and Converting to ASICs (High Level Process)

Europa Design process – Starting with FPGAs and  
converting to ASICs

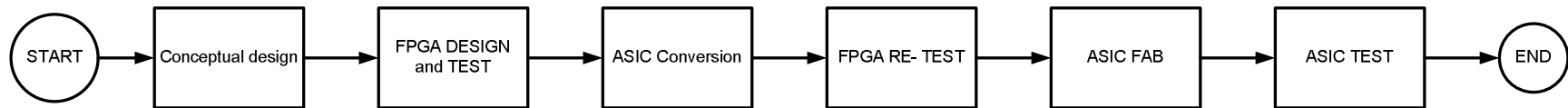


FIG 2: Starting with FPGAs and Converting to ASICs (Detailed Process)

